

CS 113 – Computer Science I

Lecture 23 – Sorting

Adam Poliak

04/18/2023

Announcements

- HW08:
 - Due 04/21 - will be released by Friday
 - Inheritance and interfaces – fully autograded
- HW09:
 - Due 04/28
 - Building a fancyArray class – fully autograded

Midterm

- Overall class did well:
 - Median 79% (Midterm 1: 77%)
 - Mean 71% (Midterm 1: 68%)
- When grading we'll lower the maximum
 - Instead of being out of 77, the denominator will be lower
- You can make a regrade request on Gradescope

Course evaluations

What do you see as the major strengths of Adam Poliak in this course? What areas do you see for improvement in instruction and/or in content?

How prepared were you to take this course? What courses, if any, would you have found useful to take before this course? Is this course listed at the appropriate level?

How did Adam Poliak effectively create an accessible and inclusive course experience? What areas do you see for commendation and/or improvement in the instructor's attention to accessibility and inclusivity?

Would you recommend this course, as taught by Adam Poliak, to other students? Why or why not?

Title	Unique ID	Instructor	Enrollments	Responded	Response Rate
Computer Science I - bmc.CMSC.B113.001.SP23	bmc.CMSC.B11 3.001.SP23	Adam Poliak	32	2	6.25%
Computer Science I - bmc.CMSC.B113.00A.SP23	bmc.CMSC.B11 3.00A.SP23	Adam Poliak	32	4	12.5%

Outline

- Announcements
- Sorting - two algorithms
 - BubbleSort
 - SelectionSort

Who do we care about sorting?

Makes searching faster!

Efficient [sorting](#) is important for optimizing the [efficiency](#) of other algorithms (such as [search](#) and [merge](#) algorithms) that require input data to be in sorted lists. Sorting is also often useful for [canonicalizing](#) data and for producing human-readable output.



ChatGPT's response:



Sorting is a fundamental operation in computer science because it is a common and essential task in many applications. Sorting refers to arranging a collection of elements in a particular order, such as numerical, alphabetical, chronological, or any other custom order. Here are a few reasons why we care about sorting in computer science:



1. Searching: Sorted data is easier to search. Searching an unsorted list requires examining every element in the list, which can be slow and inefficient for large data sets. In contrast, searching a sorted list can be done using efficient algorithms like binary search that require examining only a logarithmic number of elements.
2. Data analysis: Sorting can help in data analysis. Sorting data can help identify patterns and relationships that may not be immediately apparent in an unsorted data set. For example, sorting a list of sales data by date can reveal trends and seasonality.
3. Optimization: Sorting is often a pre-requisite for other algorithms, such as finding the median or calculating the mode of a data set. Sorting can also be used as a building block for other algorithms, such as merge sort or quick sort, which are used in many computational applications.
4. Database management: Sorting is important in managing databases. Databases often contain large amounts of data that need to be sorted in order to perform operations such as searching, filtering, and data analysis.

In summary, sorting is an essential operation in computer science that helps in searching, data analysis, optimization, and database management. It is a basic building block for many algorithms and is crucial for efficient and effective computation.

Covered in data structures, next semester!

Sorting

How might we sort the list of numbers below.
Can we come up with an algorithm?

0	1	2	3	4	5
10	4	3	0	11	8

Bubble Sort

Compare two adjacent items, and swap if needed

Repeat until largest item is at the back

Repeat process until done

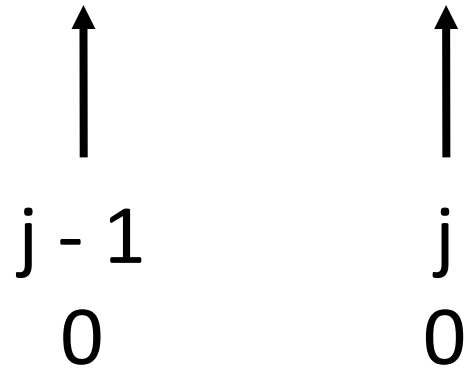
Bubble Sort

0	1	2	3	4	5
10	4	3	0	11	8

What do we do first?

Bubble Sort

len = 6

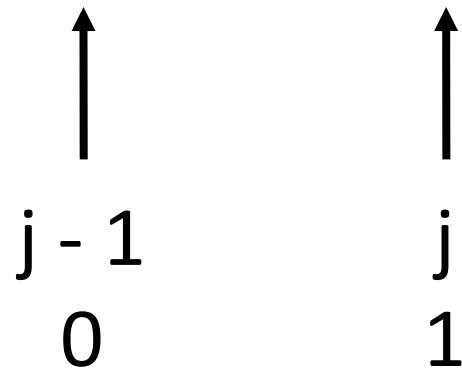


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6

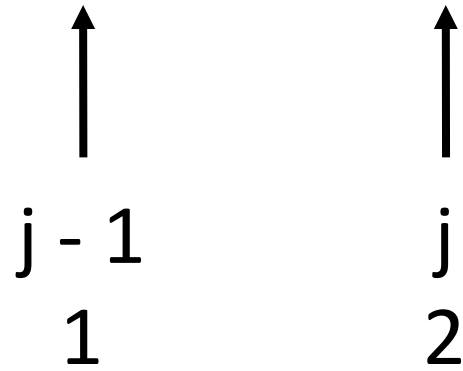


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6

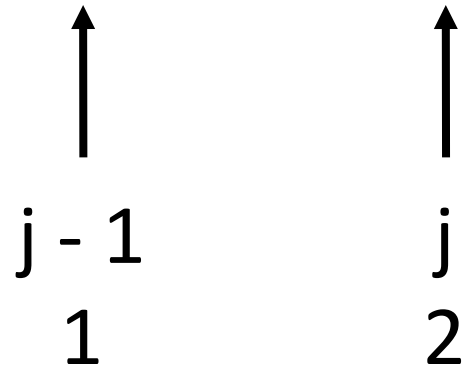
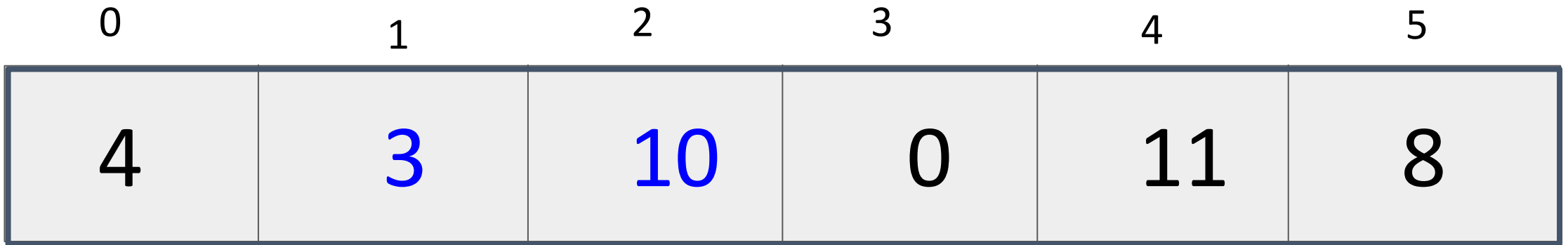


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6

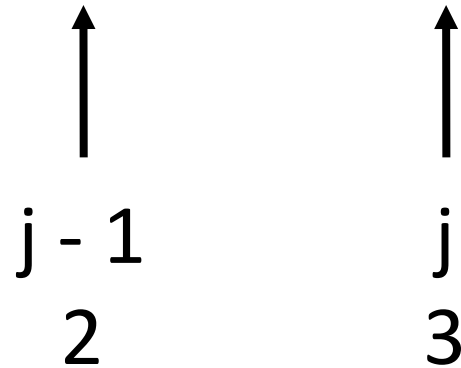


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6

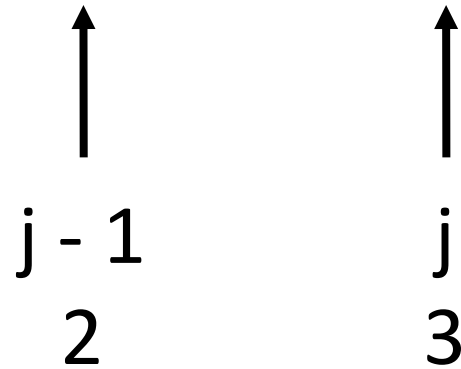
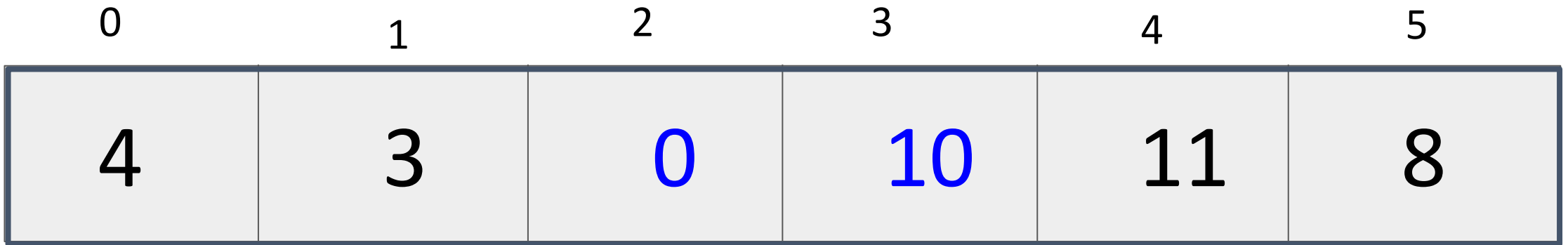


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6

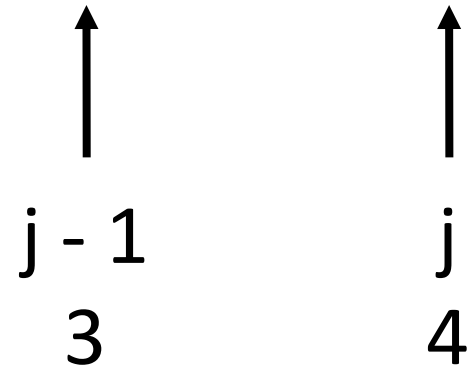


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6

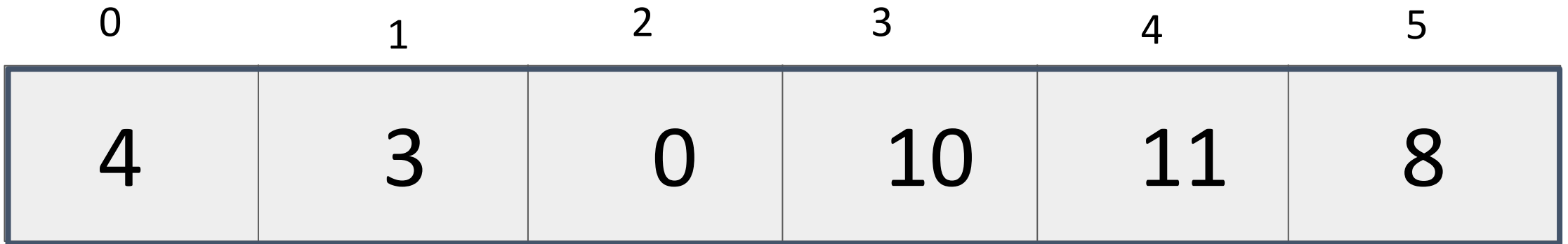


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6



↑
j - 1
4

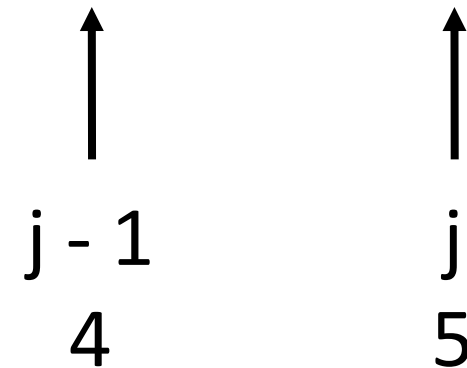
↑
j
5

Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 6



Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 5



↑
j - 1
0

↑
j
1

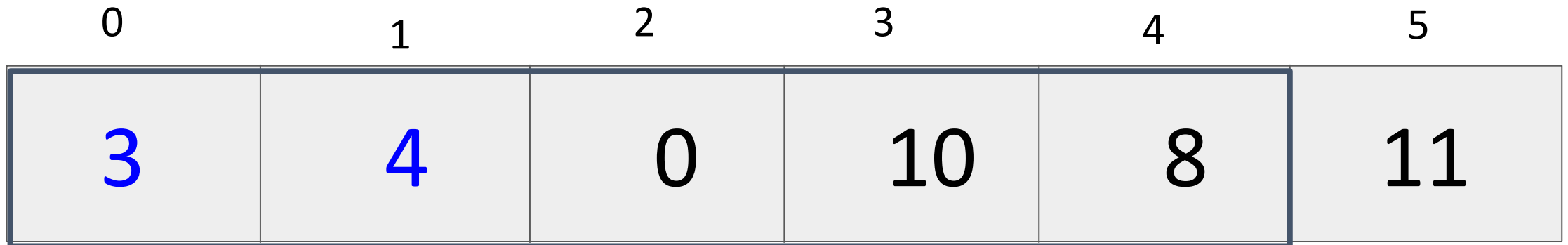
Last element has
largest element!

Reset and compare pairs with shorter list!

What next?

Bubble Sort

len = 5



↑
j - 1
0

↑
j
1

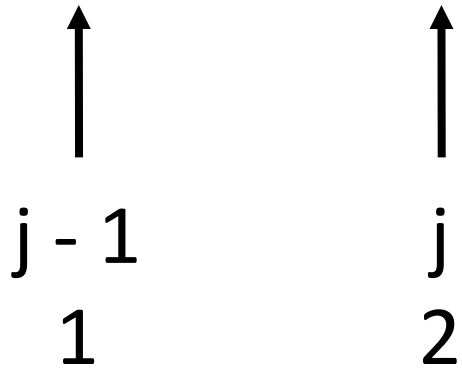
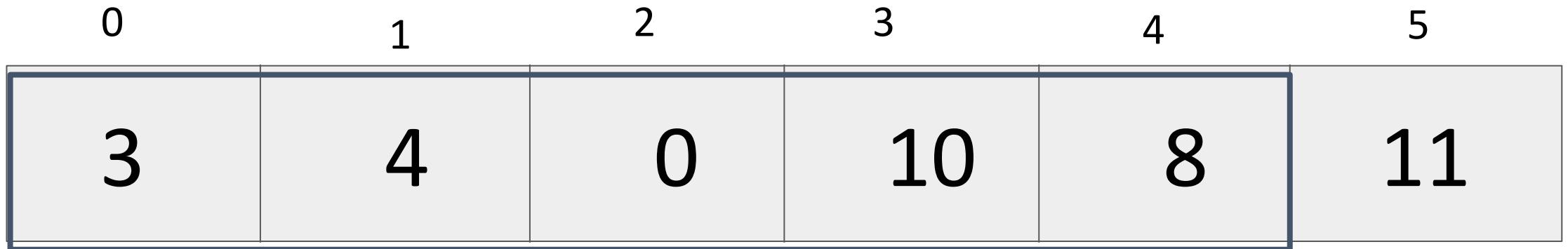
Last element has largest element!

Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 5

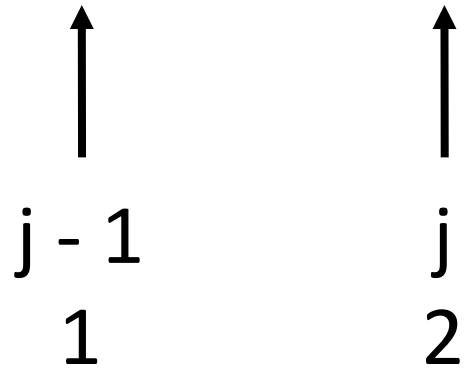
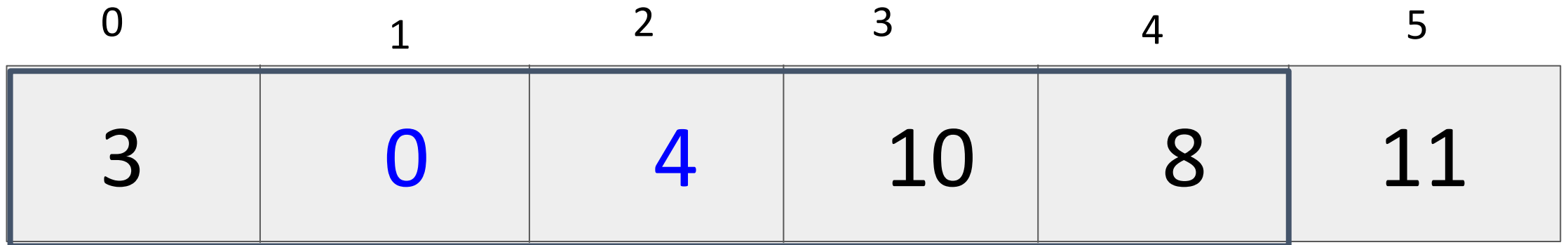


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 5

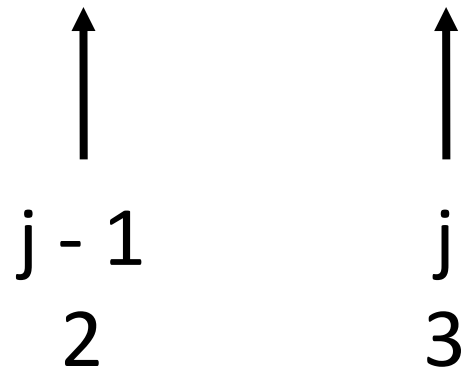
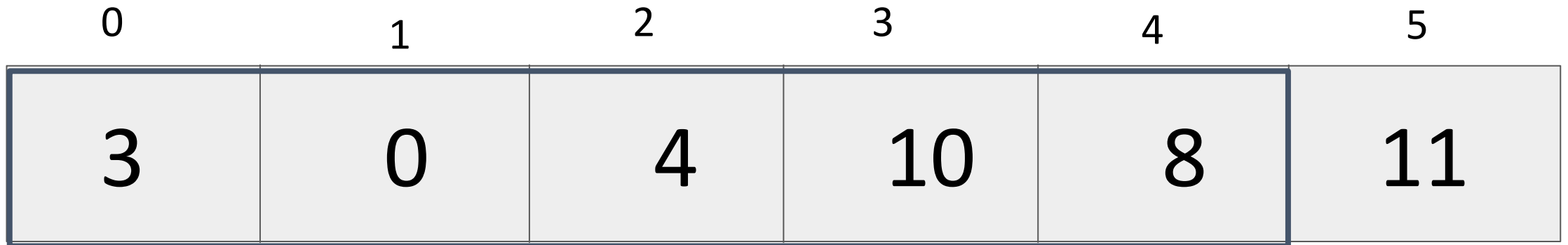


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 5

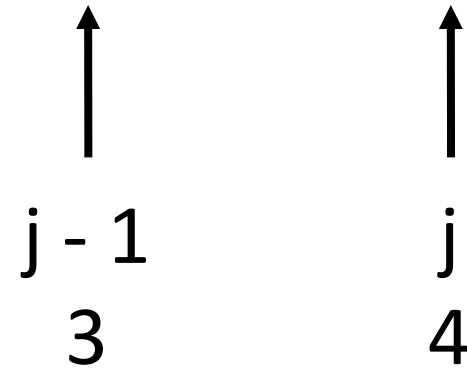
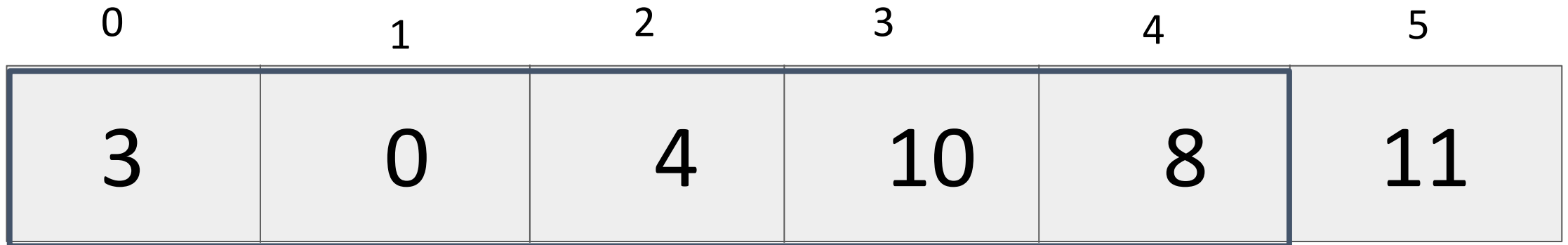


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 5

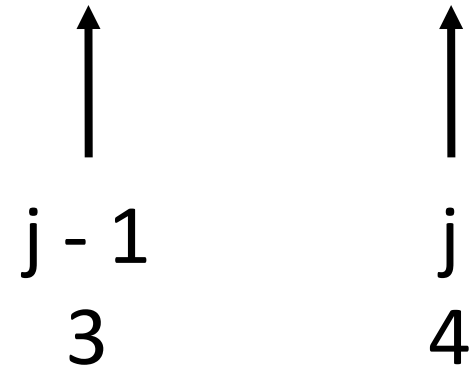
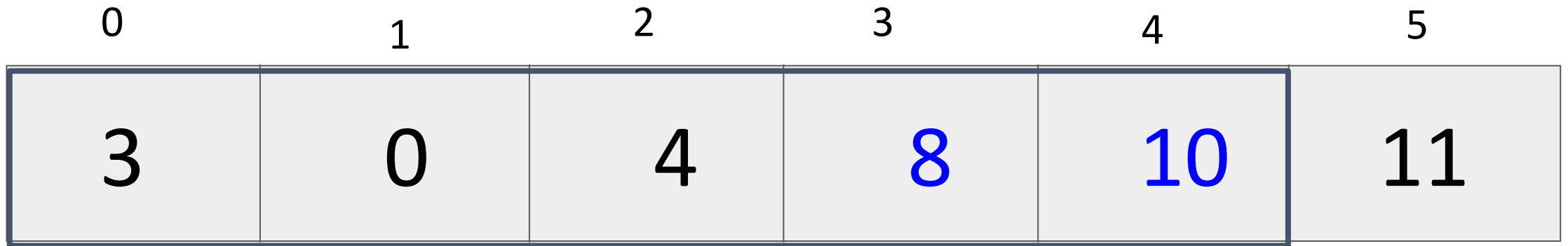


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 5

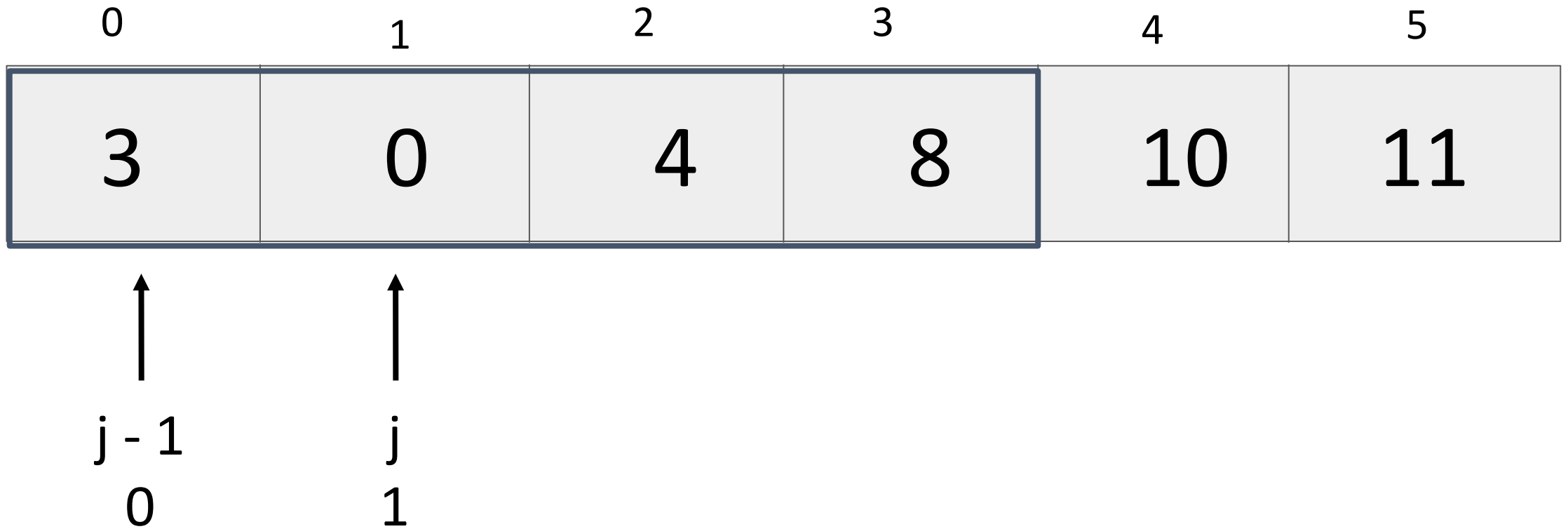


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 4

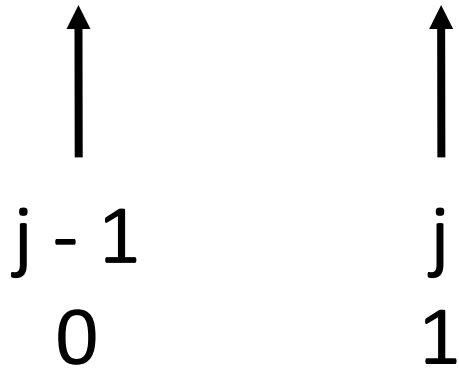
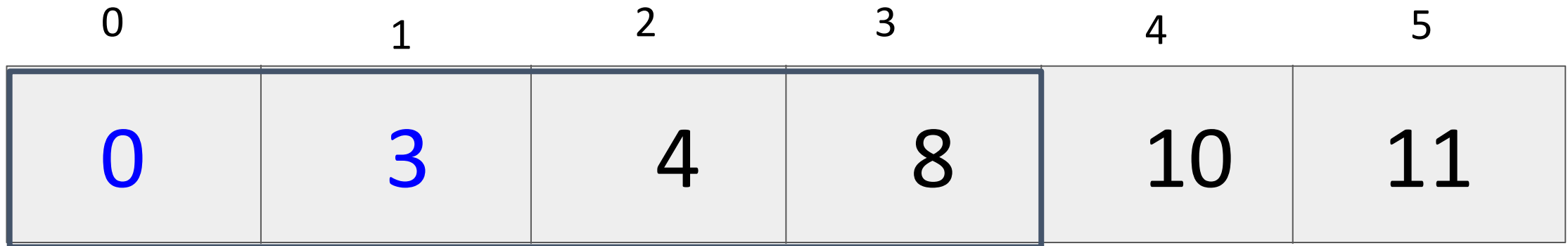


Reset and check pairs with shorter list

What next?

Bubble Sort

len = 4

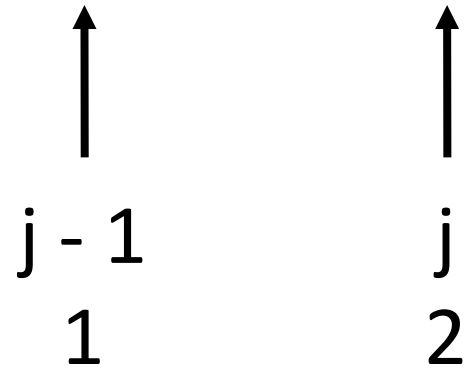
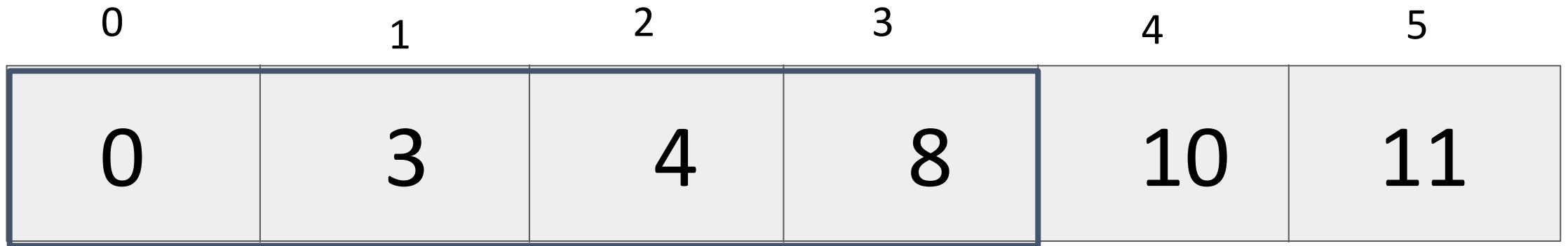


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 4

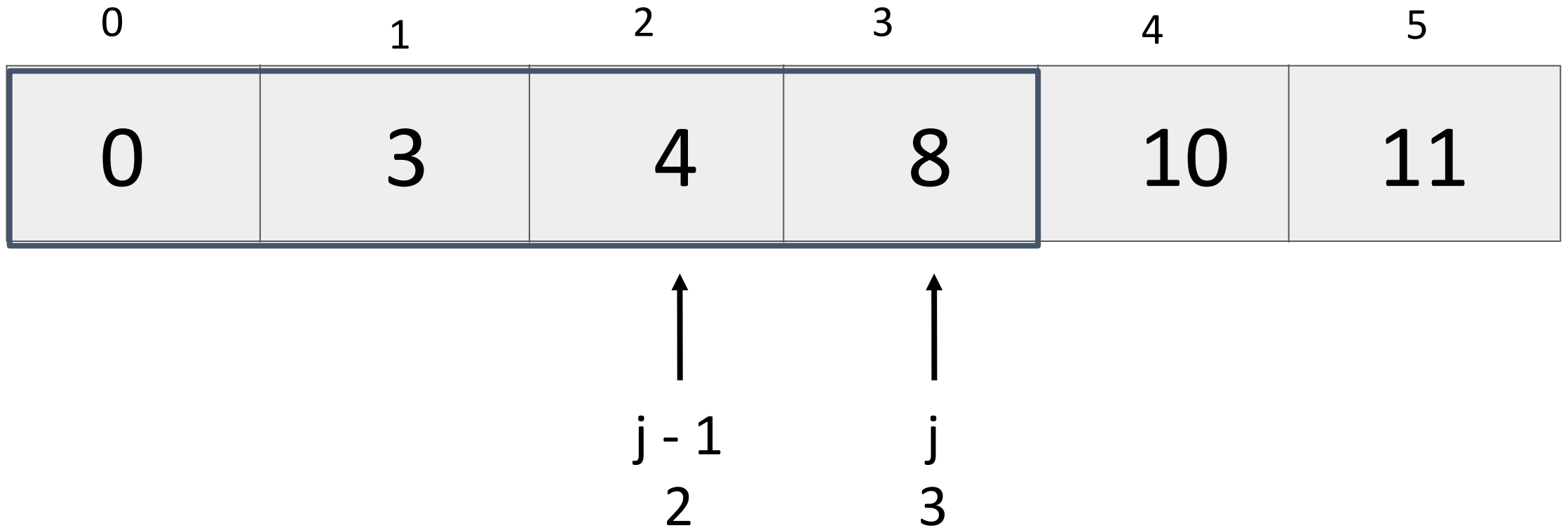


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 4

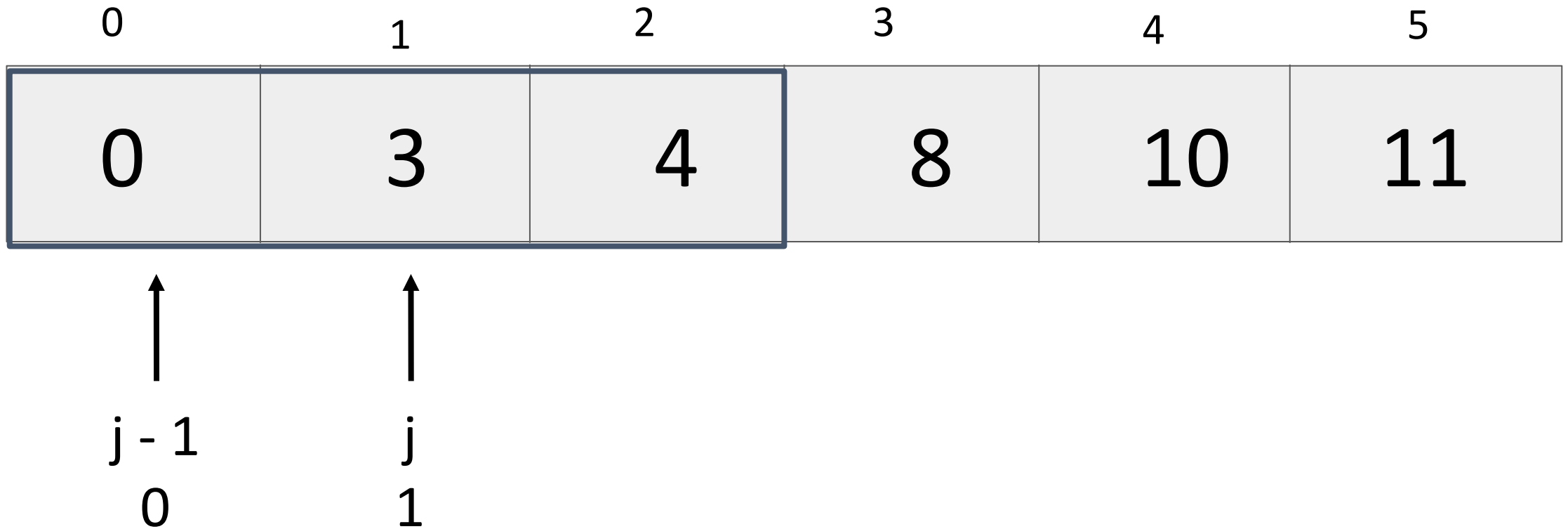


Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 3

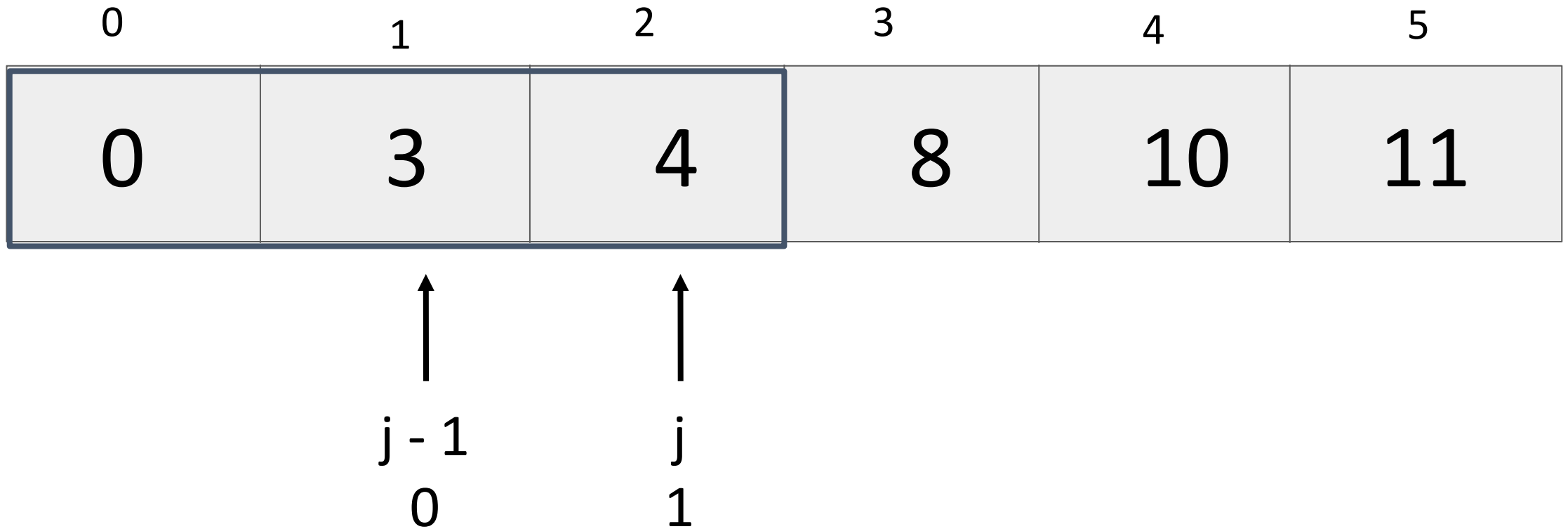


Reset; Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 3

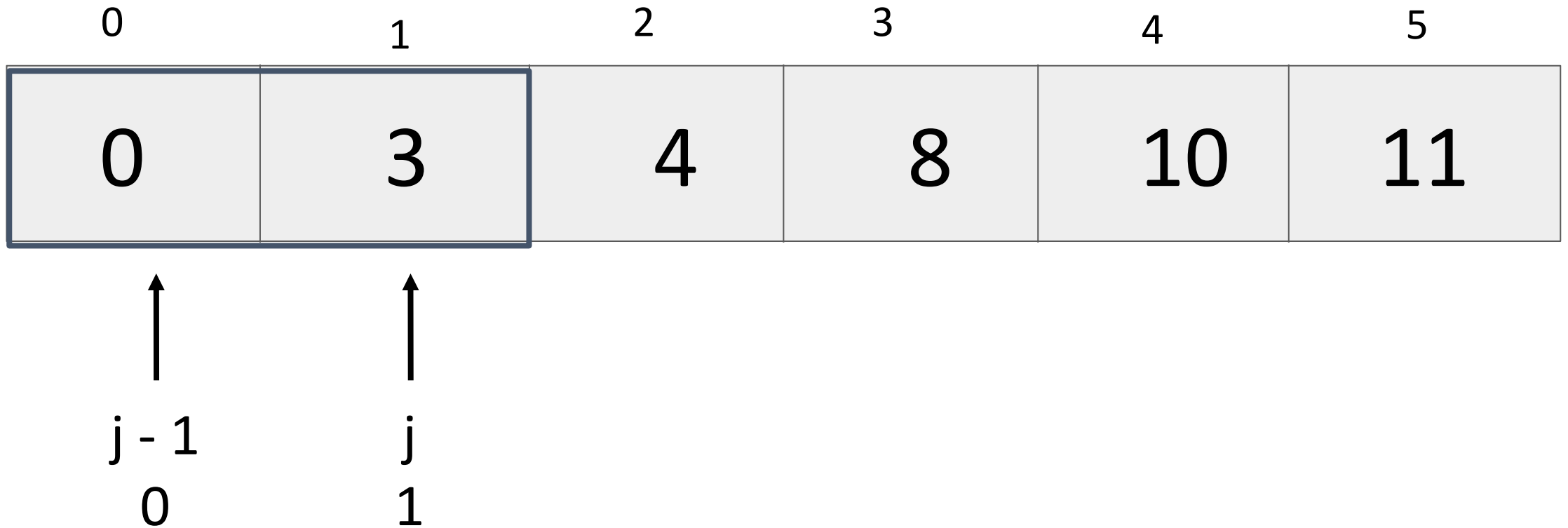


Reset; Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

len = 2



Reset; Compare $j-1$ and j ; Swap if $L[j-1] > L[j]$

What next?

Bubble Sort

Idea: bubble highest values to the end of the list; Check a shorter sublist each time

`bubbleSort(L):`

```
    for len in range(len(L), 1, -1):
```

```
        for j in range(1, len): # bubble
```

```
            if L[j-1] > L[j]:
```

```
                swap(j-1, j, L)
```

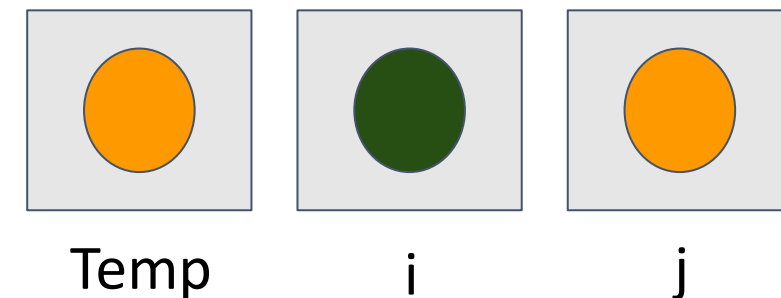
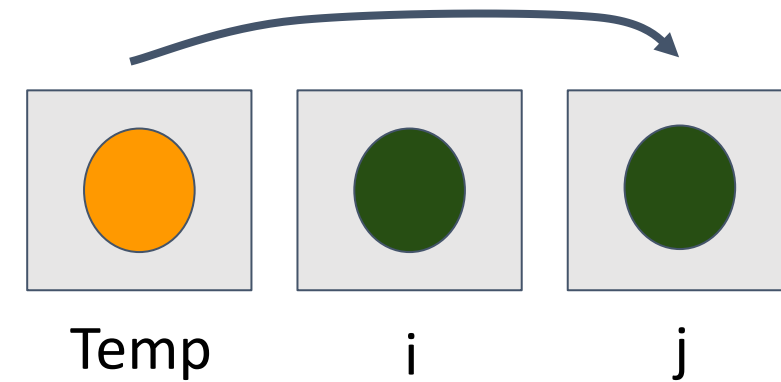
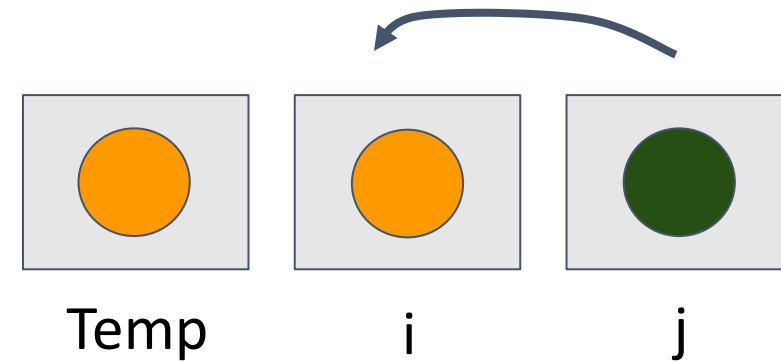
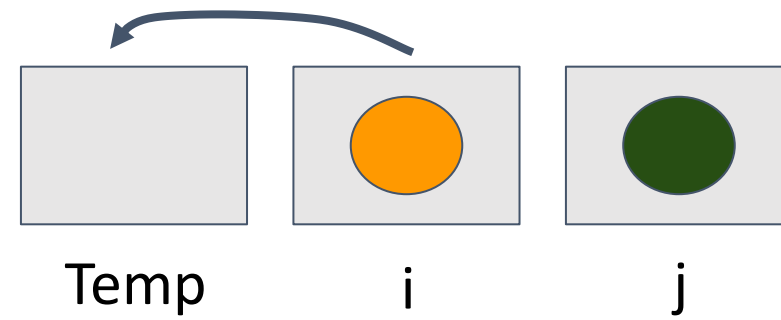
Bubble sort

swap(i, j, L):

temp = L[i] # step 1

L[i] = L[j] # step 2

L[j] = temp # step 3



Selection Sort

Selection sort

Repeatedly find the smallest item and put it at front of list

selectionSort(L):

 for startIdx in range(len(L)):

 minIdx = findMinimum(startIdx, L)

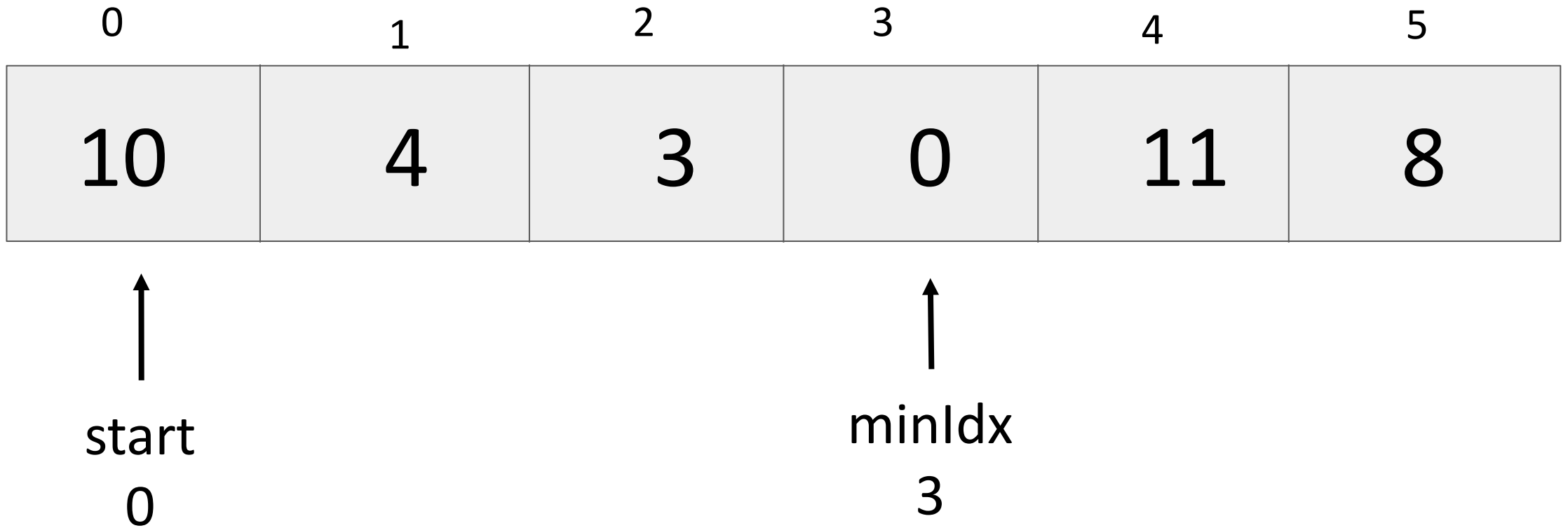
 swap(startIdx, minIdx, L)

Selection Sort

0	1	2	3	4	5
10	4	3	0	11	8

What do we do first?

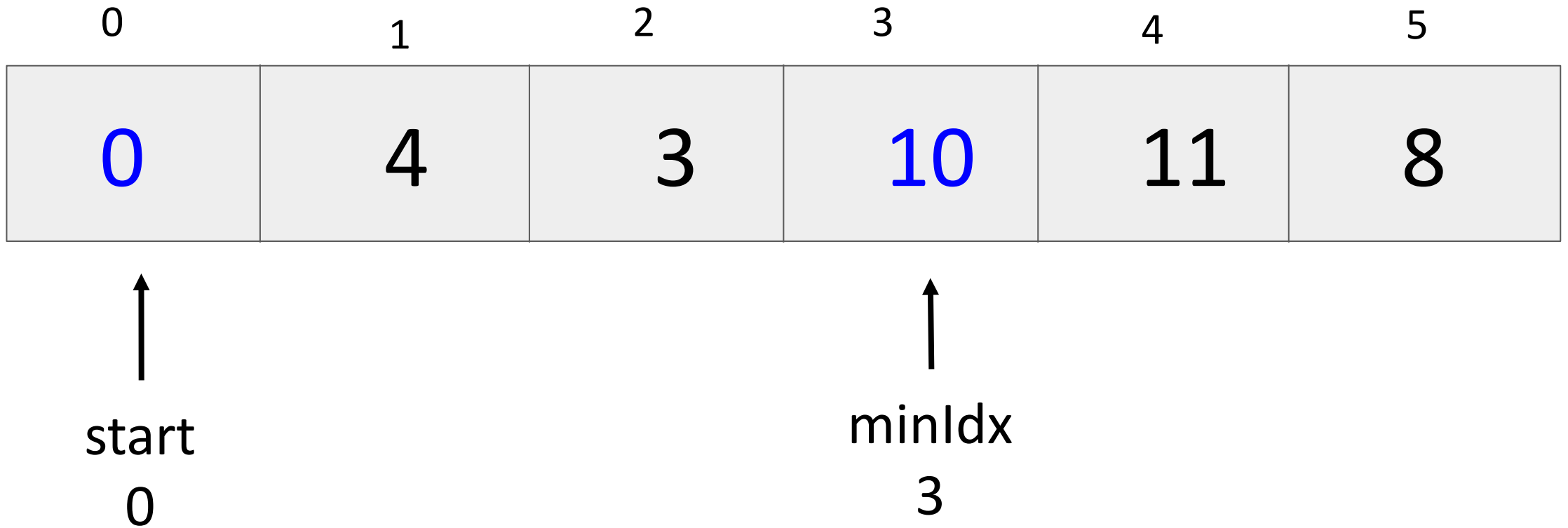
Selection Sort



Find minimum element idx between start to end

What next?

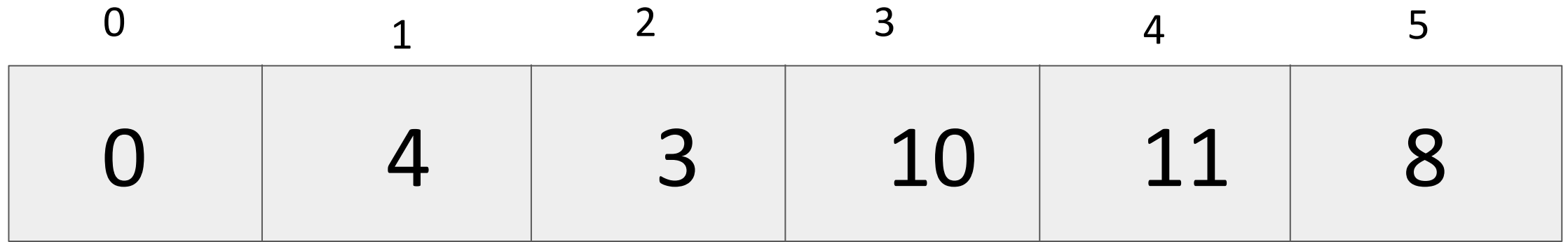
Selection Sort



Swap the elements at start and minIdx

What next?

Selection Sort

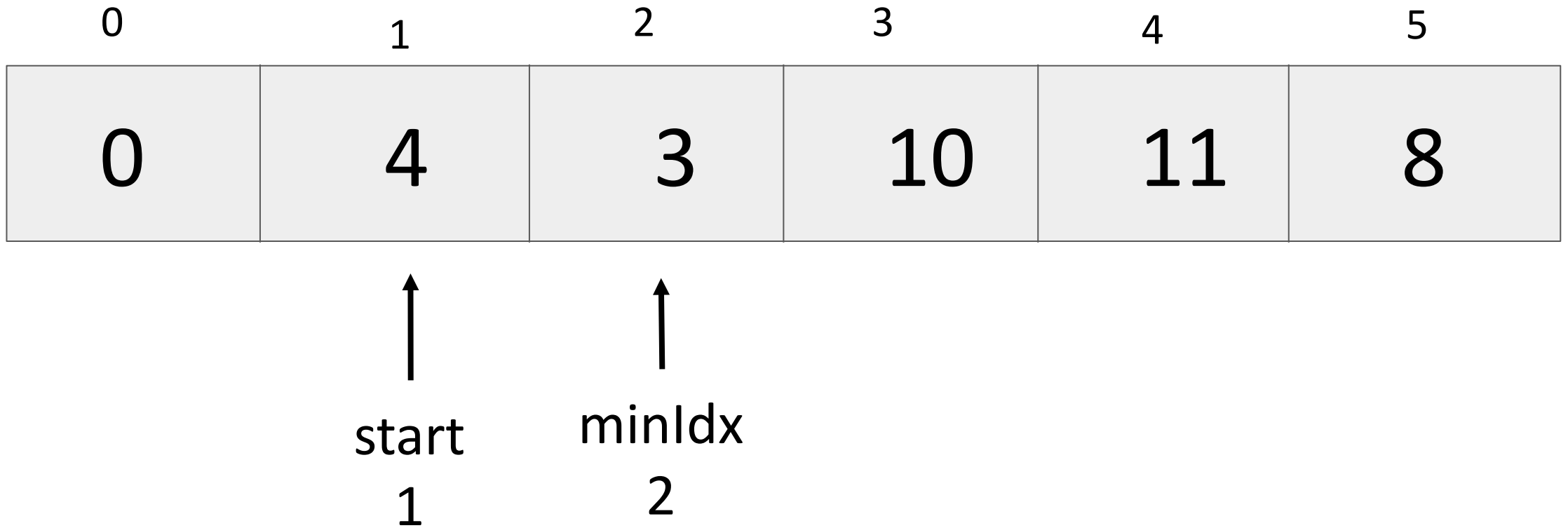


↑
start
1

Decrease the interval.

What next?

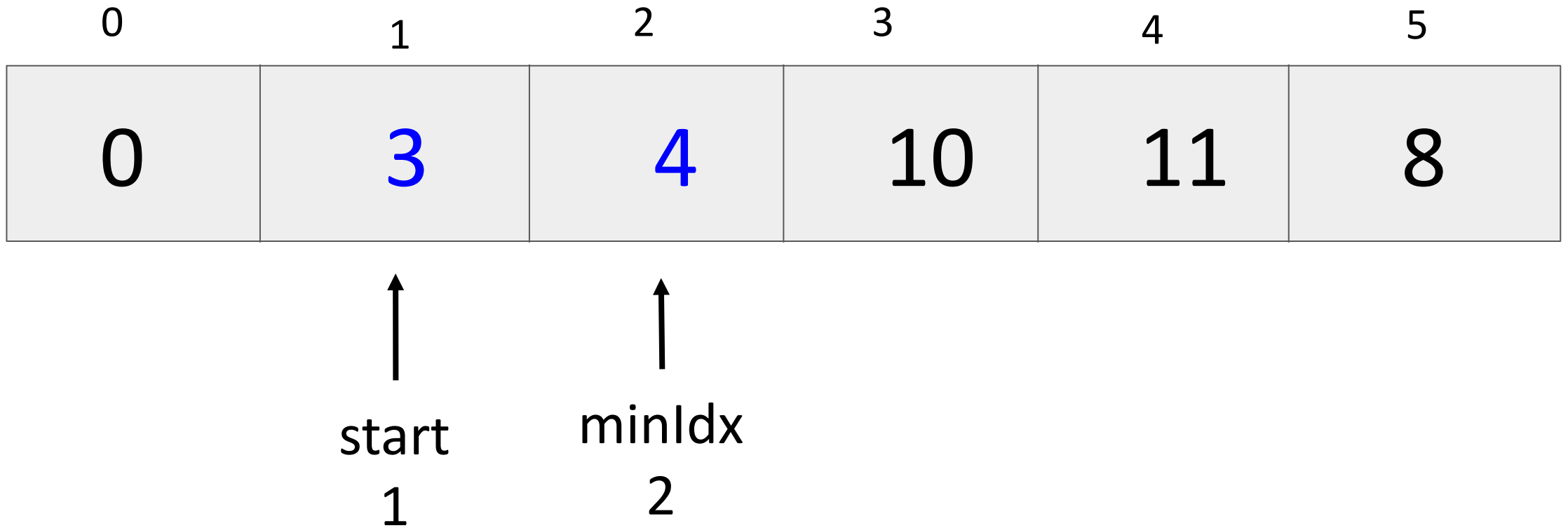
Selection Sort



Find minimum element between start to end

What next?

Selection Sort



Swap the elements at start and minIdx

What next?

Selection Sort

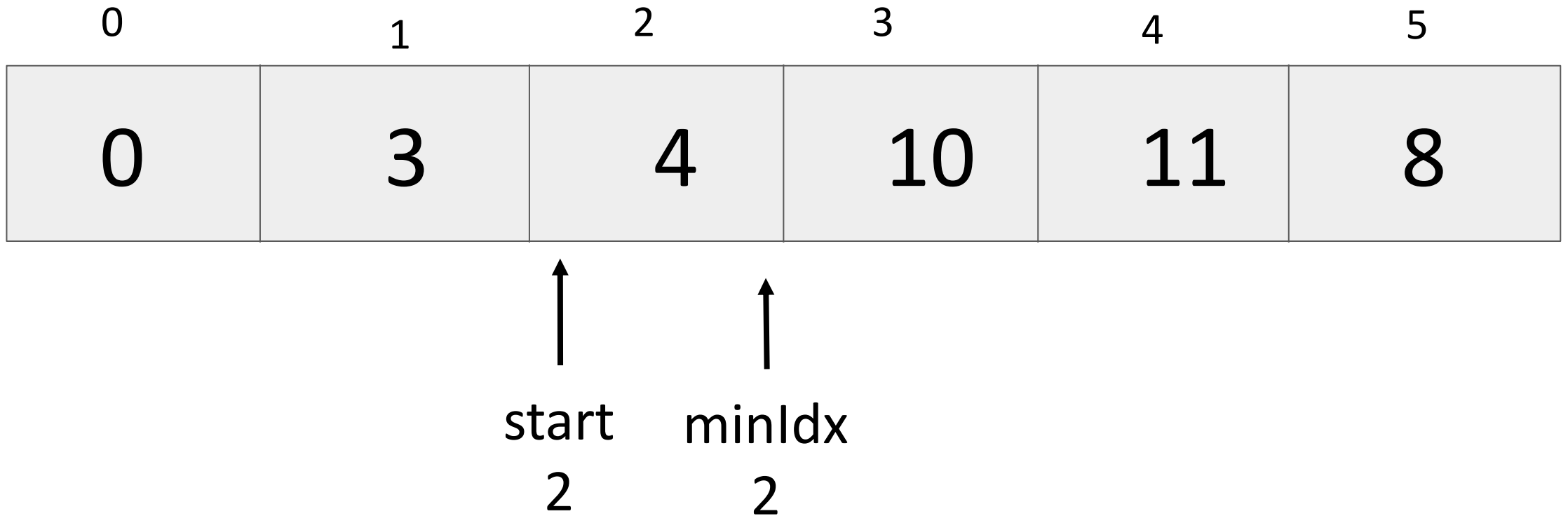


↑
start
2

Decrease the interval.

What next?

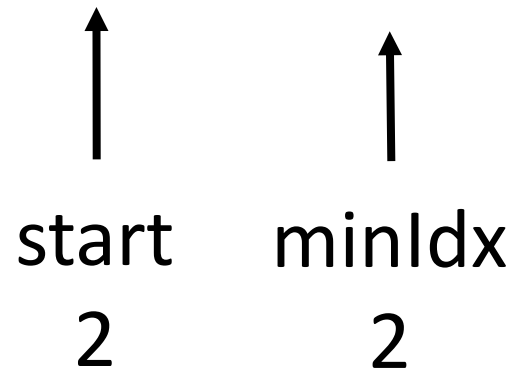
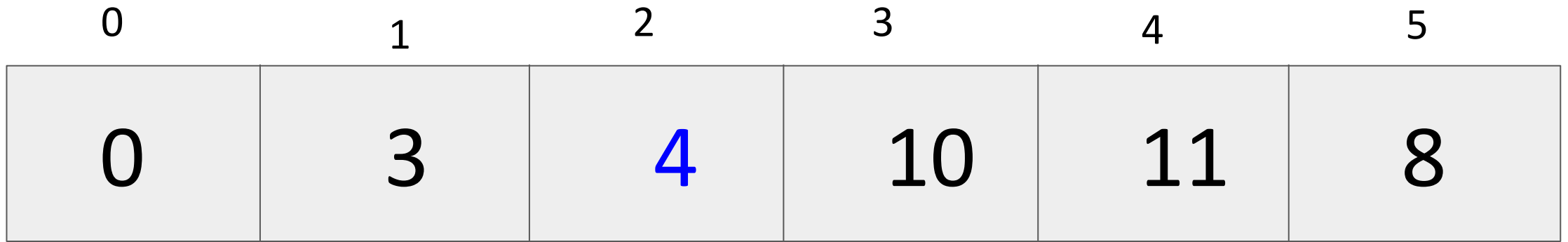
Selection Sort



Find minimum element idx between start to end

What next?

Selection Sort



Swap the elements at start and minIdx

What next?

Selection Sort

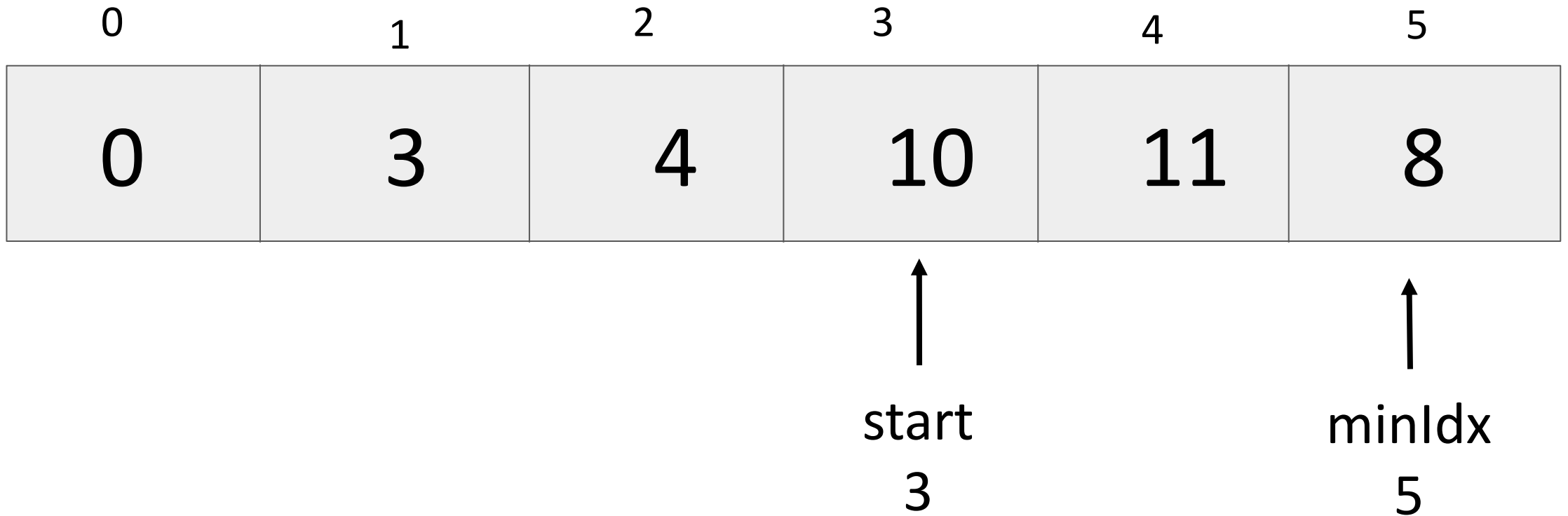
0	1	2	3	4	5
0	3	4	10	11	8

↑
start
3

Decrease the interval.

What next?

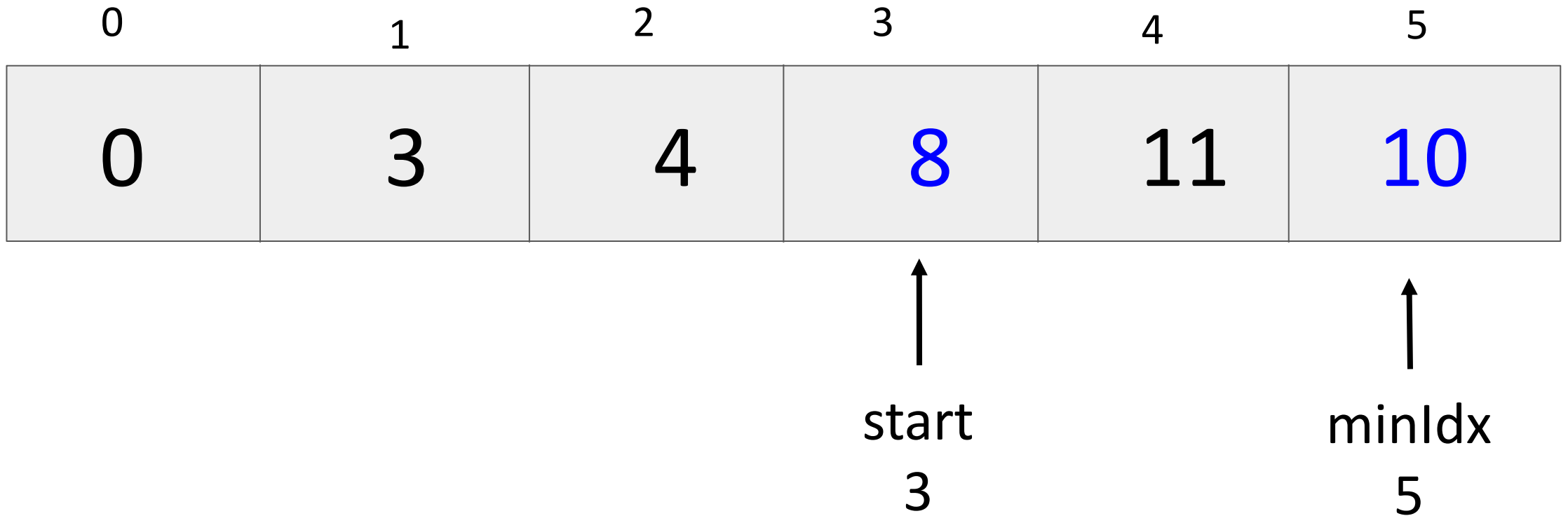
Selection Sort



Find minimum element idx between start to end

What next?

Selection Sort



Swap the elements at start and minIdx

What next?

Selection Sort

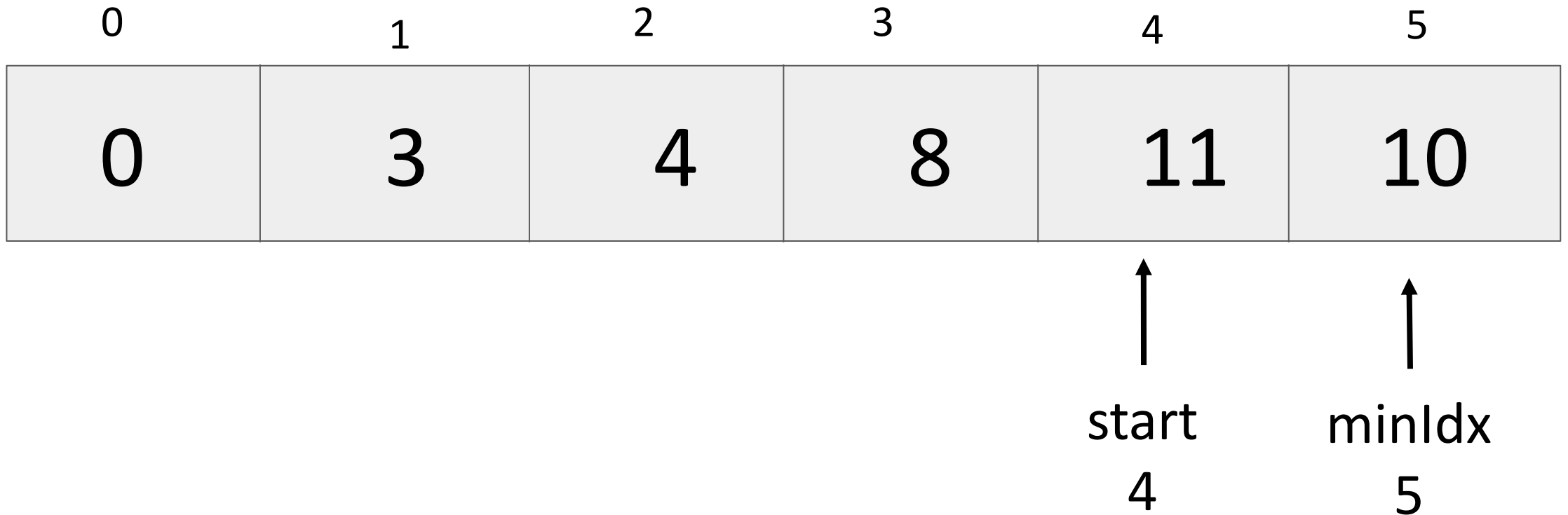
0	1	2	3	4	5
0	3	4	8	11	10

↑
start
4

Decrease the interval.

What next?

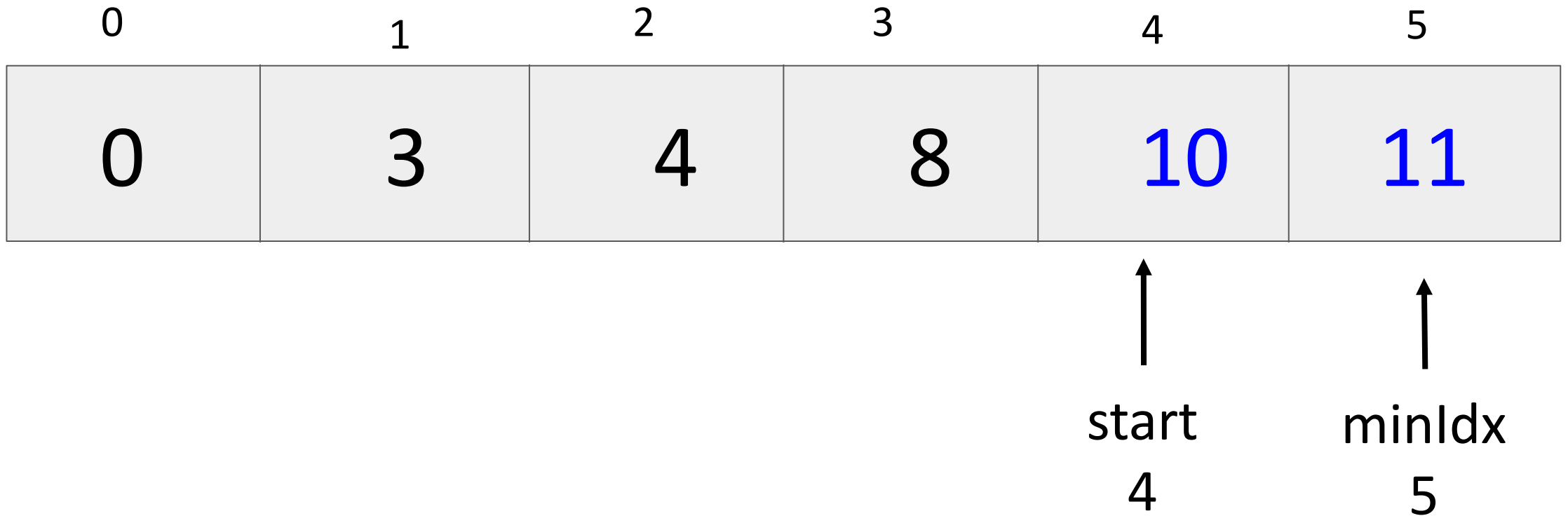
Selection Sort



Find minimum element idx between start to end

What next?

Selection Sort



Swap the elements at start and minIdx

What next?

Selection Sort

0	1	2	3	4	5
0	3	4	8	10	11

↑
start
5

Decrease the interval.

We're done!

Selection sort

```
findMinimum(startIdx, L):
```

```
    minIdx = startIdx
```

```
    for i in range(startIdx, len(L)):
```

```
        if L[i] < L[minIdx]:
```

```
            minIdx = i
```

```
    return minIdx
```

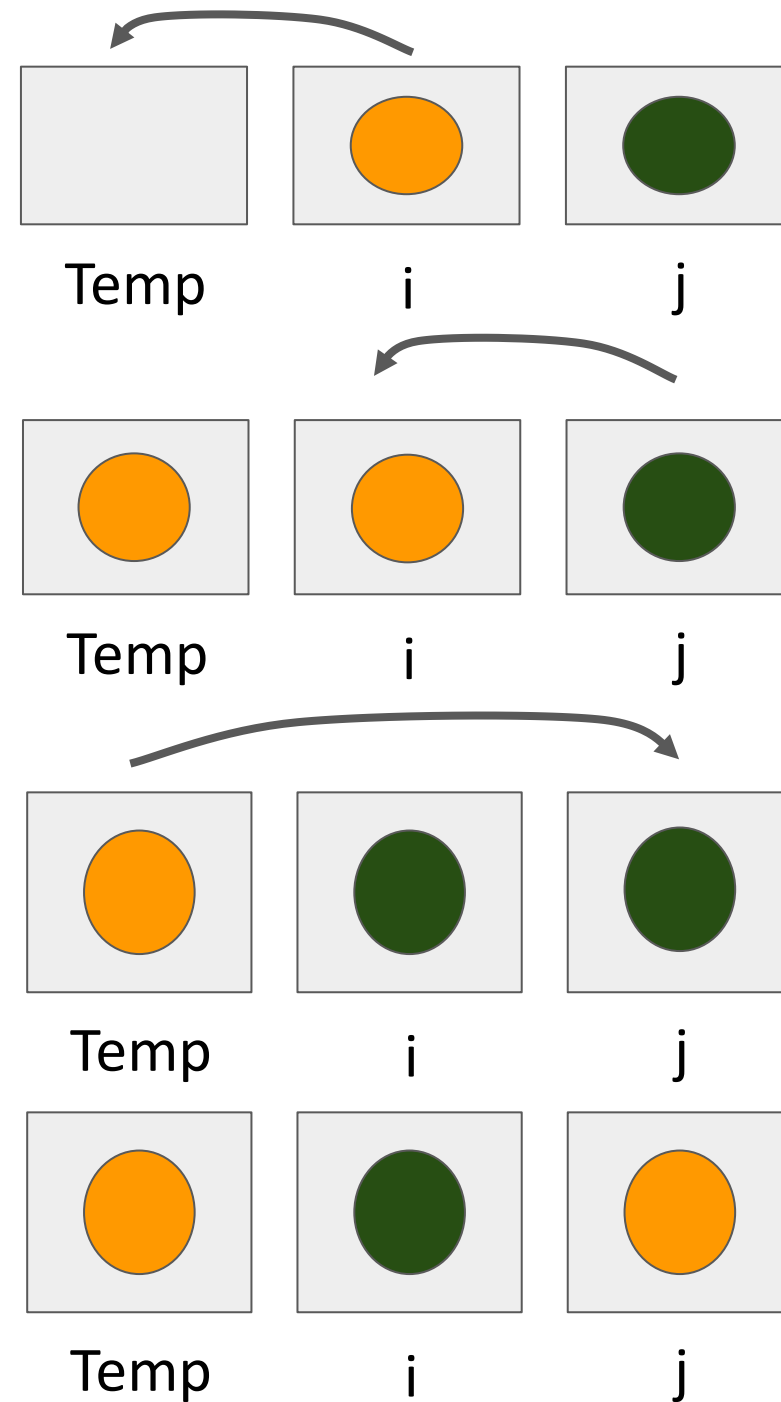
Swap

swap(i, j, L):

temp = L[i] # step 1

L[i] = L[j] # step 2

L[j] = temp # step 3



Selection sort and Bubble sort are $O(N^2)$

